Natural Discourse Hypothesis Engine

Susanna Cumming
Department of Linguistics
Campus Box 295
University of Colorado
Boulder, Colorado 80309

Abstract

As text generation systems get more sophisticated and capable of producing a wider syntactic and lexical range, the issue of how to choose among available grammatical options (preferably in a human-like way) becomes more pressing. Thus, devisers of text generation systems are frequently called upon to provide their own analyses of the discourse function of various kinds of alternations. In this paper, I describe a proposed research tool which is designed to help a researcher explore and analyze a natural-language "target text" in order to determine the contextual factors that predict the choice of one or another lexical item or grammatical feature. The project described in this paper is still at the drawing-board stage; I welcome suggestions about ways it could be changed or expanded to fulfill particular analytic needs.

Theoretical preliminaries

While some aspects of a natural-language text are determined by the nature of the information a speaker wishes to convey to a hearer, there are many more aspects that seem to be determined by certain cognitive needs that the hearer has. Speakers tailor their output to the informational resources and deficiencies of the hearer in several ways: by adjusting the amount of information they make explicit, by arranging new information relative to old information in a maximally helpful way, and by giving special marking to information that may be difficult for the hearer to access for a variety of reasons. It is these strategies that give rise to the wide variety of syntactic and lexical resources of any natural language for saying the "same thing" in different ways. We can call the relation between lexico-grammatical features and the speaker's communicative goals in choosing those features the "discourse functions" of the features.

For any particular alternation, then, the best predictor of the speaker's choice should be a model of the cognitive state of the hearer. Unfortunately, neither human speakers nor computer systems have direct access to the hearer's mind. But linguists have long realized that we do have access to a fair approximation of an important subset of the information the hearer possesses at a given point in a discourse: namely the text which has been produced up to that point. ([Chafe 1987] and [Givón 1983] are two contemporary expressions of that principle.) And in fact we can make fairly good predictions of lexico-grammatical choices based on inferences that come from the nature of the preceding text. For instance, a referent that has been referred to in the previous clause is likely to receive minimal coding (a pronoun or zero, depending on syntactic considerations). But this principle can be overridden by the presence of other factors that interfere with the accessibility of the referent — e.g. a paragraph break or another competing referent — resulting in the use of a full noun phrase. Or, to give another example, a speaker is likely to use special syntax (such as the "presentative" or "there is..." construction) to introduce a referent that will be prominent in the following discourse; so a hearer is likely to have easier access to a referent introduced in that way than one that has been introduced "casually", e.g. in a prepositional phrase. Therefore, subsequent references to a referent that has been introduced with a presentative are more likely to be pronouns than noun phrases.

These are all factors that can be discerned in the preceding text and are taken into account by speakers as affecting the nature of the hearer's expectations. Therefore under these perturbing circumstances the speaker can decide to use a fuller reference, e.g. a proper name or noun phrase. Figure 1 illustrates the relation between the discourse produced by the speaker, the hearer's mental state, and the speaker's model of the hearer. In real face-to-face interaction, the hearer can influence the speaker's model of her or him in more direct ways — e.g. by verbal and non-verbal indications of agreement, understanding, protest, or confusion. But this two-way channel is available neither to human writers nor to computer text generators.

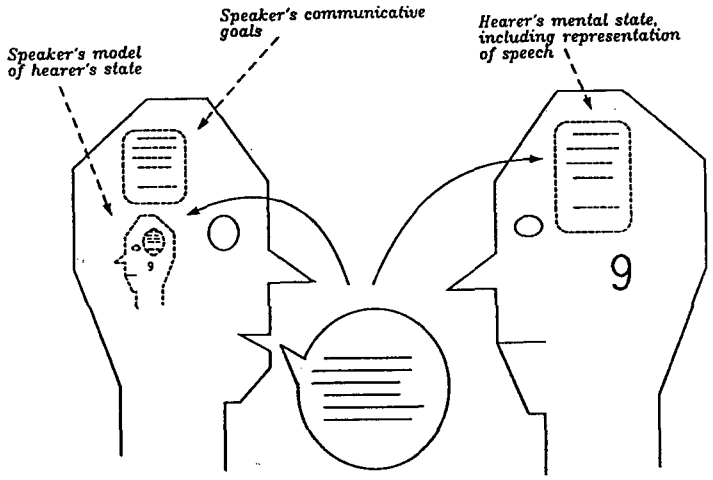

Figure 1: A model of communication.

The fact that we can draw inferences from preceding text about the hearer's state justifies the methodology common in functional linguistics whereby the linguist investigates the function of a particular morpho-syntactic alternation by attempting to discover correlations in natural texts between that alternation and various features of the context. This method is also likely to be the one which will lead to the best results for text generation specialists, since prior text is one information source any computational system has easy access to as a guide in making future decisions.

There are, of course, currently available several large-scale databases of English texts of various kinds, some of which have been provided with various amounts and kinds of coding (from word-class tagging to syntactic analysis). However, for many kinds of discourse work these databases have not been useful. On the one hand, such databases are often not available in languages other than English; on the other, the coding that is provided assumes an analysis which may not meet the needs of a particular user. In fact it is often the case that aspects of the syntactic analysis depend on a functional analysis having already been done; so a pre-coded database may contain assumptions about the answers to the same questions it is supposed to be helping to solve. The tool described here is designed for the user who is satisfied with a relatively small amount of data, but wants to have total control over the analysis of the corpus.

Currently, functional linguists often enter their text data into commercial relational database tools, such as DBase and Paradox. However, such tools have been designed with other kinds of applications in mind, and therefore there are many properties of natural language texts which these tools can only capture by awkward or indirect means. Specifically, natural language texts, unlike e.g. client or sales lists, are simultaneously *linear* (order matters) and *recursively hierarchical*. Thus, the query languages available with such database products are generally good at extracting the kind of information from texts which is least useful to linguists, but require considerable ad hoc programming to extract the kind of information most relevant to our needs. The tool proposed here, conversely, should answer most easily the questions a discourse analyst wants to ask most frequently.

These problems in representing certain crucial aspects of natural language texts computationally using off-the-shelf tools have sent many linguists back to marking up xeroxed texts by hand with colored pencils. The approach outlined here is intended to combine the advantages of the colored-pencil technique (spontaneity, flexibility, and direct involvement with the whole text) with the advantages of computational techniques (quick and painless identification, compilation, and comparison of various features of text units).

Description of the tool

The tool proposed in this paper will aid in the generation of hypotheses concerning the discourse function of a lexico-grammatical feature (or combination of features) by allowing the researcher to isolate instances of that feature and view them in relation to various aspects of its discourse context. When the researcher has arrived at and stated a working hypothesis about which features of the context are most relevant to predicting the targeted feature, the system will be able to test the hypothesis against the data and provide feedback by displaying both predicted and non-predicted occurrences of the target feature. Further refinements can then be made until the fit between the researcher's hypothesis and the actual text is as good as possible. The hypothesis can then be integrated into a text generation system with some assurance that the lexico-grammatical choices made by the system will approximate those made by a human with a similar text-production task.

In order for the tool to be able to recognize and compare various text features — including relatively covert information such as the reference of noun phrases and the mood of a sentence as well as relatively overt cues such as lexical items and structure — the user must first annotate the text, indicating a) its hierarchical structure, and b) a set of features (attribute/value pairs) associated with each constituent. The annotation will be largely manual (though aspects of it can be automated, as will be explained), because we want the tool itself to be neutral as

to the theoretical assumptions inherent in any particular analysis. Thus, it will be possible to use this tool to test among other things the comparative predictive usefulness of various possible analyses of a text.

Using the tool will have the following phases:

1. *Annotation:* mark constituents, and for each constituent indicate its category membership and any other desired features. The amount and type of annotation is entirely up to the user; it will never be necessary to include any more detail than is relevant for the current analysis. Constituents may be anything from words to paragraphs or episodes.

2. *Hypothesis formation:* view the target feature and context in various arrangements.

3. *Hypothesis testing:* state a hypothesis in terms of a pattern in the context that predicts the occurrence of the target feature.

4. *Hypothesis refinement:* make changes in hypothesis, annotation, or both and retest against the same data; extend analysis by testing against different data.

1. Annotation

An example text showing segmentation and sample feature specification associated with three of the constituents is given in figure 2.

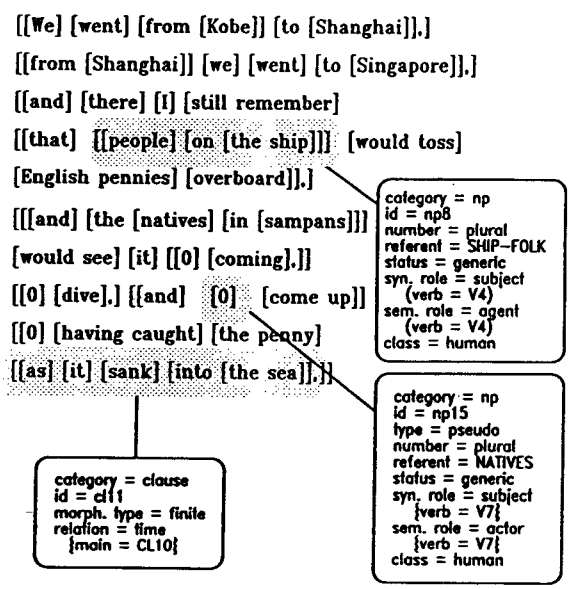

Figure 2: Annotated text.

Annotation will have the following steps: segmentation, category specification, further feature specification, and dependency specification. Furthermore, "pseudo-constituents" may be added and specified at this stage, as explained below.

1. *Segmentation*: Mark off a stretch of text to be considered a unit. Initially, allowable units must be continuous and properly contained inside other units; but eventually it would be nice to allow two kinds of violations to this principle, namely discontinuous constituents and overlapping constituents.

2. *Category specification:* Specify the category of the constituent. The category will be stored as an attribute-value pair (with the attribute "category") like other features. At the time of category specification, the constituent will be assigned a unique ID, also an attribute-value pair (with the attribute "id").

3. *Further feature specification:* Specify further attribute-value pairs. In order to ensure consistency the user will be prompted with a list of a) attributes that have previously been used with that category, and b) values that have previously been associated with that attribute. The user may select from these lists or enter a new attribute or value.

4. *Dependency relations:* Some kinds of relations between constituents are best treated by specifying dependency relations between the two items, i.e. by providing a labelled pointer linking the two. Syntactic analyses differ from each other in what kinds of relations are treated as constituency relations (e.g. the relationship of sister nodes in a tree) and what kinds are treated as pure dependency relations.

5. *"Pseudo-constituent" insertion:* For some purposes, it may be desirable to allow the insertion of "pseudoconstituents" in the text, i.e. phonologically null "elements" that can be inferred from the surface text, or boundaries which aren't marked by any explicit linguistic element (such as "episode boundaries" or possibly the beginning and end of the text). Examples of null elements which can be tracked in discourse as if they were overt include "zero pronouns", inferrable propositions, and elements of frames or schemas. Pseudo-constituents can be inserted directly into the text like actual constituents, but they should have distinctive marking in their feature specification to indicate their abstract status; in the examples I have distinguished them by the feature "type = pseudo".

Figure 3 illustrates a possible implementation (using pop-up menus) of the process of feature specification.

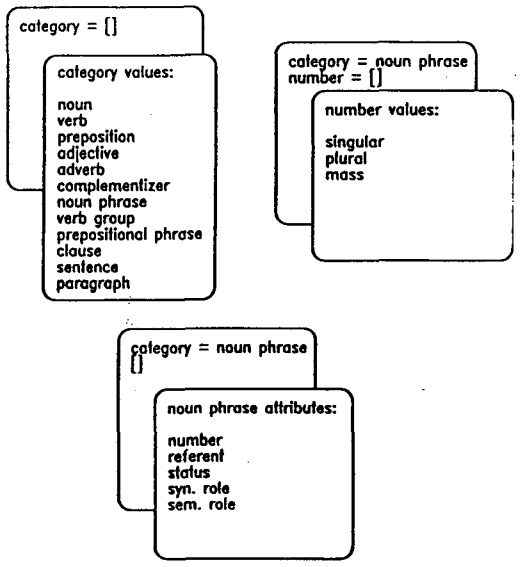

Figure 3: Feature specification.

Queries

Having annotated a text to the desired degree, it is then possible to start fishing around for an analysis of the distributional characteristics of the feature or combination of features of interest to the analyst (henceforth "target pattern"). Experience has shown that the most useful ways to do this is to get an idea of the overall frequency of the phenomenon, and then to "eyeball" the target pattern in various kinds of context. Thus, the tool should have the following capabilities:

1. *Counting:* Count the number of instances of the target pattern.

2. *Highlighting:* Display the whole text, with constituents which match the target pattern highlighted by color or a distinctive font.

3. *Extraction:* Extract constituents which match the target pattern into a file of the same type as the text file. It will then be possible to print out the output file, or to subset it further by performing additional highlighting or extraction queries on it.

Basic pattern specification. These operations require a format for describing the target pattern. I propose an interface which will allow the user to open a "query window", in which he or she can enter a partial specification of one or more constituents to be matched against the text, using the same entry techniques as are used to enter, edit, and annotate actual text. This approach has the advantage of letting the user make use of what he or she already knows about the annotation process to specify the target pattern, and furthermore it allows the identification of highly complex patterns in an intuitive way.

A pattern specification will contain the following kinds of elements:

1. Actual structure (brackets), text, and feature bundles.

2. Variables which stand for text or features, used to indicate relationships between parts of the pattern, or to specify to parts of the pattern to be acted on (counted, highlighted, extracted).

3. Various kinds of wildcard elements which will match unspecified strings of structure or text.

4. Operators (and, or, not) which specify relations between features, text, variables, or whole patterns.

It should be possible to save a query in a file for future editing and re-use. This avoids repetitive typing of the same query, while permitting refinement and adjustment of queries that don't give the desired results the first time. It also allows the creation of a series of related queries by copying and slightly altering a prototype.

Variables (indicated here by strings of alphanumerics starting with "#") are used to specify text or elements of feature bundles for reference elsewhere, either to be matched or acted upon. A text variable is interpreted as matching all the text in the constituent that contains it (including text in embedded constituents). For instance, the following two queries would match constituents containing two elements of the same category conjoined by "and", as in "Jill and Liza" or "sang off-key and danced the waltz". The (a) version would highlight the whole conjoined phrase; the (b) version would highlight each of the conjoined phrases, but not "and". (The dots are wildcards, explained below. In these and the following examples, feature specifications are indicated in small italics; literal text is enclosed in quotes.)

(1a) [#txt [..]*cat=#c* "and" [..]*cat=#c*]
highlight #txt

(2b) [[#txt1 ..]*cat=#c* "and" [#txt2 ..]*cat=#c*]
highlight #txt1 and #txt2

An analysis of the types of queries I anticipate wanting to make reveals that a fairly diverse set of wildcards is necessary. Structure wildcards match elements of structure, i.e. combinations of brackets. Text and features are ignored by these wildcards (see below). Two types of structure wildcards are provided. The first, composed of periods, stops at the first instance of the pattern in the string; the second, composed of exclamation marks, finds all instances of the pattern.

1st match	All matches	
.	!	Depth: a series of all right or all left brackets
..	!!	Constituents: a series of matching left and right brackets (and their contents)
...	!!!	Any material

The following examples illustrate the use of structure wildcards:

(2) [[#X]*cat=np* ..]*cat=clause*

X matches a top-level NP which is the first constituent of a clause. It matches "Jill" in "Jill ate the peach", but nothing in "On the next day Jill ate the peach" (because the first top-level element is a prepositional phrase, not a noun phrase).

(3a) [. [#X]*cat=np* ..]*cat=clause*
(3b) [! [#X]*cat=np* ..]*cat=clause*

X matches an NP which is part of the first constituent in a clause, no matter how deeply it is embedded. In the sentence "On the day after the fire Jill ate the peach", both (a) and (b) will match "the day after the fire", and the (b) version will also match "the fire".

(4a) [.. [#X]*cat=np* ..]*cat=clause*
(4b) [!! [#X]*cat=np* ..]*cat=clause*

X matches the first NP which is a top-level constituent of the clause; in (b), it matches all NPs which are top-level constituents of the clause. In the sentence "On the day after the fire Jill ate the peach", both versions will match "Jill"; (b) could also match "the peach" if the sentence was given a "flat" structural analysis (without a verb phrase).

(5a) [... [#X]*cat=np* ..]*cat=clause*
(5b) [!!! [#X]*cat=np* ..]*cat=clause*

X matches the first noun phrase in a clause (no matter where it is); (b) matches every noun phrase in a clause. In "Much to the surprise of her neighbors, Jill ate the peach", (a) will match "the surprise of her neighbors", while (b) will additionally match "her neighbors", "Jill", and "the peach".

Text/feature wildcards are used both to match actual text and to match elements of feature bundles (attributes and values).

(nothing)	Matches any text/feature or no text/feature
@	Matches no text (the absence of text)
*	Matches any text, attribute, or value

The following examples show how these wildcards could be used:

(6) [#X @ [ ]*cat=clause*]*cat=np*

X matches a noun phrase whose first element is a clause. It would match "that he came" in "that he came surprised me", but not in "the fact that he came surprised me".

(7) [#X]*ref=**

X matches any constituent that has a referent specified.

Figure 4 illustrates one way windows and menus could be used to simplify the query process. The query window is divided into two panes, a "pattern pane" and an "action pane". The various kinds of query elements can be either typed in or selected from a set of pull-down menus. This query is designed to extract noun phrases which have the property of "referential ambiguity" in the sense of [Givón 1983]: referents which have been referred to in the immediately previous clause, but where the previous clause also contains another referent of the same semantic class. (An example would be "Mary saw Jill, and *she* ran off"; *she* is referentially ambiguous, and in fact we would expect a full noun phrase rather than a pronoun in this kind of context.)

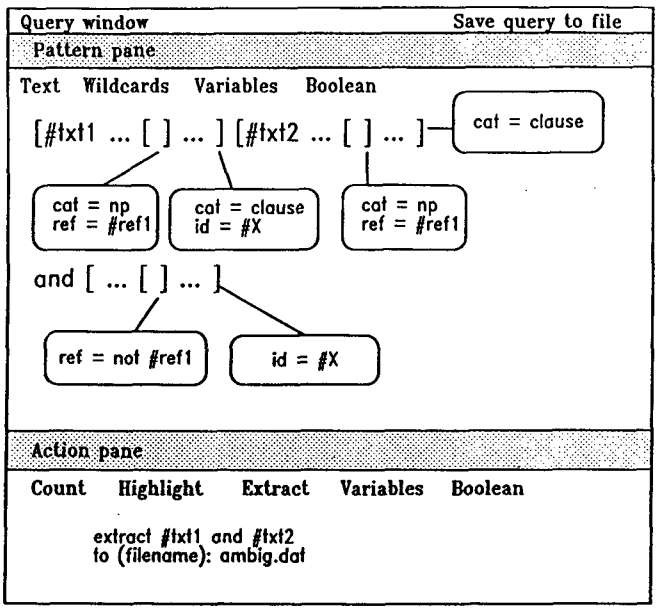

Figure 4: A query.

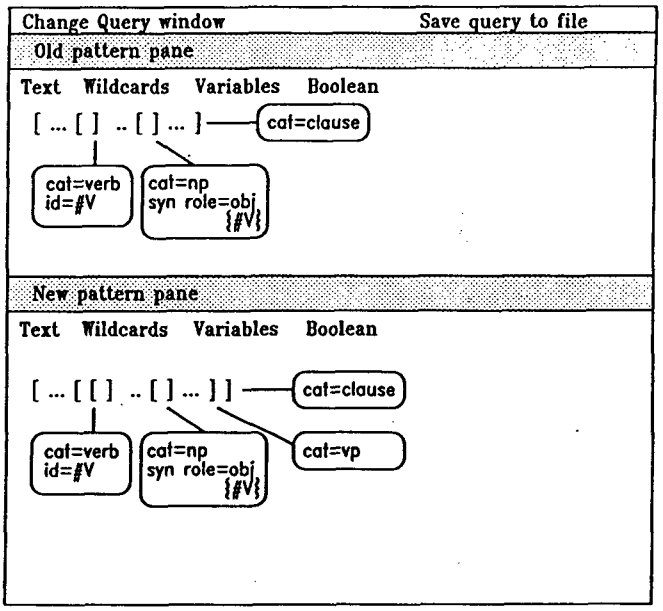

Figure 5: A change query.

Using queries to change annotation

Extensive experience with feature-coding of text data has shown that there is a large number of features which are predictable from the identity of lexical items and/or combinations of other features. In these cases it is very convenient to be able to partially code a text and to fill in the redundant codes by using queries. Similarly, it is sometimes desirable to use redundant information in an annotated text to change or delete features when you change your mind about the best coding scheme. Furthermore, if it is possible to insert structure boundaries on the basis of patterns found in the text, it is even possible to perform some structure insertion automatically, giving the user in effect the possibility of using queries to automatically parse the text where this can be done reliably.

he most straightforward way to implement this is to allow the user to specify two patterns, an "old" pattern and a "new" pattern. The query illustrated in figure 5 creates a verb phrase constituent wherever a verb is followed by a noun phrase which is its object.

Distance queries. When text is viewed as a window to the cognitive processes of the speaker, it becomes apparent that the distance between e.g. two mentions of a referent or two clauses referring to the same event can be important. This is because the evidence suggests that the cognitive accessibility (to the hearer) of a referent "decays" over time, so that speakers need to do more work to accomplish reference to something that has not been mentioned recently (cf. [Givón 1983, Chafe 1987]; the explanation presumably has to do with general properties of short term memory storage). For this reason, it is desirable to have a way to measure the text distance between two items. There are various ideas "in the air" about what the appropriate units for measuring distance are (e.g. clauses, intonation units, conversational turns), and different measures are clearly appropriate for different text types; so it is desirable to define distance queries so that the user can specify the unit to be counted as well as the beginning and ending patterns (the two items whose separation is being measured). The result of a simple distance query is naturally an integer; but where the beginning and ending patterns are general, the result should be a list of integers which can then be averaged. Finally, in order to make it easy to reference distance information in future queries, it should be possible to save the distance result as a value of a feature with a user-specified attribute.

Figure 6 illustrates a query which computes "lookback" (i.e. the distance between two NPs with the same referent) for pronouns, using the clause as a unit. It averages the

lookback value, so that average pronoun lookback can be compared with e.g. average full noun phrase lookback.

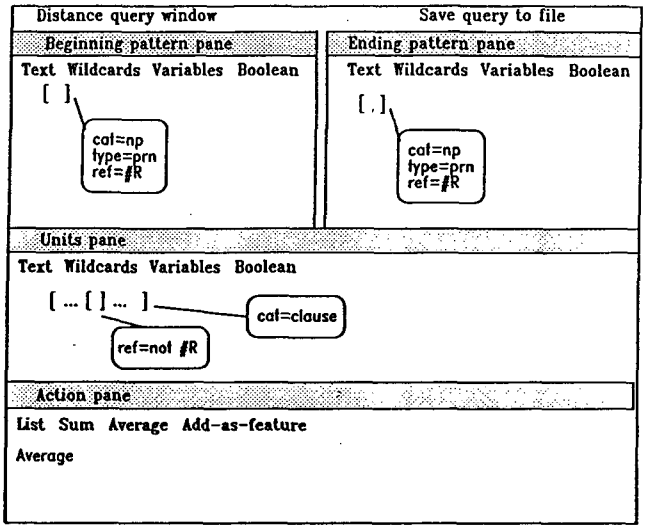

Figure 6: A distance query for "lookback".

Another significant use for the distance query is to compute "mention number", that is, whether a noun phrase contains the first, second, third etc. mention of a referent. This could be accomplished by measuring the distance between a noun phrase and the beginning of the text, using noun phrases referring to the same entity as the unit of distance. Such a query is illustrated in figure 7 (which assumes that text boundaries have been marked with pseudo-constituents).

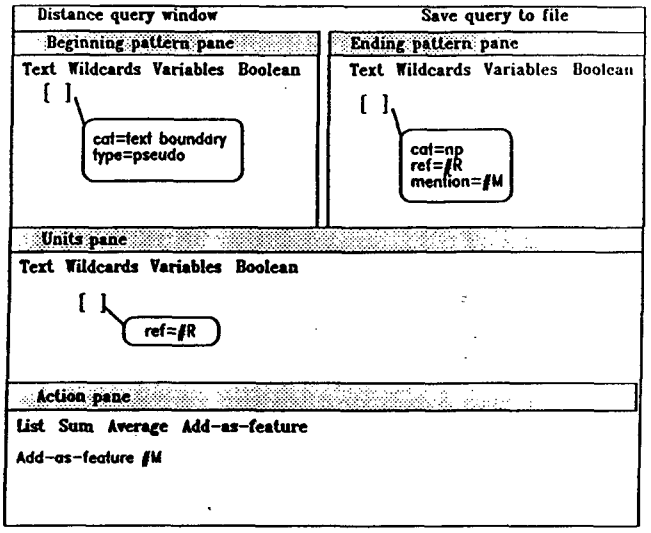

Figure 7: A distance query for "mention".

Additional components

In the future, it would be very desirable to implement the possibility of adding additional components so that all linguistic information need not be stored directly in the text. For instance, it would be nice to allow a lexicon component, which would contain partial feature specifications of lexical items (including e.g. category, number, tense, semantic class). Queries could then search the lexicon as well as the text for the presnece of features, obviating the need to manually perform highly predictable annotation. It is easy to see how currently available large-scale lexical databases could be adapted to this purpose, obviating a large amount of hand annotation. A "structurecon" could perform a similar function for larger units. These additions would be particularly useful for applications involving large amounts of rather similar text; they would be less useful for small texts where open-class lexical items and predictable structures are rarely repeated, or for lexically and structurally diverse texts.

Hypothesis refinement using queries

I will conclude with an example of the way this tool can be used to generate and test hypotheses about the way linguistic choices are made. Since the tool has not yet been implemented, I have chosen an area where previous research has given a good indication of what the answers to queries would probably be: the introduction into dis-

course of referents which are new to the hearer (in the sense of "nonidentifiable"). The following discussion draws largely on the results reported in [Du Bois 1980].
The category of nonidentifiable referent can be approximated by the feature "first mention". Therefore, the first step will be to use the method described above to automatically add the "mention" feature to each noun phrase (this procedure will assign the feature "mention = Ø" to first mentions). Then you can "eyeball" all first mentions in context by using the following query:

(8) [#X]*cat=np, mention=Ø*
highlight #X

Let's say you notice that many, but not all, first mentions have an indefinite article. You wonder what proportion of the data is accounted for by simply stating "first mentions have an indefinite article". You can determine this by means of the following series of queries:

(9) [#X "a" or "an" ..]*cat=np, mention=1*
count #X
'How many first mentions are marked with an indefinite article?'

(10) [#Y not "a" and not "an" ..]*cat=np, mention=Ø*
count #Y
'How many first mentions don't have an indefinite article?'

(11) [#Z "a" or "an" ..]*cat=np, mention=not Ø*
count #Z
'How many things marked with indefinite articles aren't first mentions?'

Suppose you discover that, although all NPs with indefinite articles are first mentions, and most first mentions have an indefinite article, there is a large number of first mentions which don't have an indefinite article. The next step might be to highlight these so you can view them in context and look for shared characteristics. For instance, you might notice that plurals don't have indefinite articles. You can then modify the original tests to check for number as well, e.g.

(12) [#Y not "a" and not "an" ..]
cat=np, mention=Ø, num=sing
highlight and count #Y

'How many singular first mentions don't have an indefinite article? Show them to me.'

The response to this query might well reveal another large class of first mentions which are actually marked with a definite article. These are referents which can be interpreted as "accessible" to the hearer by virtue of being closely associated with (e.g. a part of) something that has already been mentioned. One way to cope with these instances is to insert "pseudo-constituents" into the text at the point where the associated entity is mentioned, and rerunning the mention-number query. These references would no longer count as first mentions, and a retry of query 12 would reveal a much smaller number of exceptions.

This example could be extended, but it is hoped that it will illustrate the way such a tool could be used for quick and easy exploration of the functional attributes of linguistic forms.

Acknowledgments

This paper was improved by comments from Sandy Thompson, Randy Sparks, and an anonymous reviewer. They are not responsible for remaining faults.

References

[Chafe 1987] Chafe, Wallace L. Cognitive constraints on information flow. In Russell S. Tomlin (editor), *Coherence and grounding in discourse.* (Typological Studies in Language 11). Amsterdam: Benjamins, 1987.

[Du Bois 1980] Du Bois, John W. Beyond definiteness: the trace of identity in discourse. In Wallace Chafe (editor), *The Pear Stories*, 203-275. Norwood: Ablex, 1980.

[Givón 1983] Givón, Talmy, editor. *Topic continuity in discourse.* Amsterdam: Benjamins, 1983.